Enhance Your Java Stack Trace Using JPrintStackTrace

JPrintStackTrace is a powerful tool that enhances your Java stack trace and gives a more precise view of the problem to help you debug it easily.

Many a time, you may end up in a situation where the usual Java stack trace is of no use while debugging a problem. The stack trace may not show you the correct source code line that triggered the problem. Also, the program that you are trying to debug could be a legacy one, or you may not have access to the source code, and even if you did, re-compiling and building it may not be that easy. In such cases, you may need a tool to enhance the standard Java stack trace and help you debug and fix the problem easily.

In this article we will take a look at a tool called JPrintStackTrace (developed by the author) that can be used to get a more precise Java stack trace, without making any changes to the source code.

JPrintStackTrace

JPrintStackTrace is a Java Platform Debugger Architecture (JPDA)-based tool. It runs on a separate JVM and can connect to your program to enhance the Java stack trace. Since it works at the JVM level, you need not make any code changes in your program. The power of JprintStackTrace can be best explained with an example. Consider the Simple.java program shown in Figure 1.

Figure 1: Simple.java

```
1    public class Simple {
2           public void displayl4() throws Throwable {
3                String nullStr = null;
4
5                try {
6                     System.out.println("displayl4: nullStr: ' +
nullStr);
7                     nullStr = nullStr.trim();
8                } catch(Throwable t) {
9                     throw new Throwable("displayl4");
10               }
11          }
12
13          public void displayl3() throws Throwable {
14               try {
15                    displayl4();
16               } catch(Throwable t) {
17                    throw new Throwable("displayl3");
18               }
19          }
20
21          public void displayl2() throws Throwable {
22               try {
23                    displayl3();
24               } catch(Throwable t) {
25                    throw new Throwable("displayl2");
26               }
27          }
28
29          public void displayl() {
30               try {
31                    displayl2();
32               } catch(Throwable t) {
33                    t.printStackTrace();
34               }
35          }
36
37          public static void main(String args[]) {
38               try {
39                    Simple s = new Simple();
40                    s.displayl();
41               } catch(Throwable t) {
42                    t.printStackTrace();
43               }
44          }
45   }
```

The program tries to de-reference a null string at Line 7 in function *display14()*. Looking carefully at the sample program you can see that the exceptions are caught and a new exception object is thrown to the caller. This continues till the function *display1(),* where it is displayed to the user.

Now, let's compile and run this program as shown below.

```
javac Simple.java

java Simple
```

Here is the output:

Figure 2: Output of Simple.java

```
display14: nullStr: null
java.lang.Throwable: display12
        at Simple.display12(Simple.java:25)
        at Simple.display1(Simple.java:31)
        at Simple.main(Simple.java:40)
```

From Figure 2, you can clearly see that the Java stack trace is of no use. It does not specify the correct source code line that triggered the exception. Now you either have to change the source code to handle the exceptions correctly or you need to run the program step-by-step in a debugger to find the root cause. But these approaches are either too time consuming or not feasible for several reasons, as described earlier.

Running the *Simple.java* program

Now let us run Simple.java under the JPrintStackTrace tool. First, compile and run the Simple.java program in debug mode as shown in Figure 3.

Figure 3: Running Simple.java in debug mode

```
javac –g Simple.java

java -hotspot -Xrunjdwp:transport=dt_socket,server=y,
suspend=y,address=4000 -Xdebug Simple
Listening for transport dt_socket at address: 4000
```

From Figure 3, you can see that the *Simple* program has started in debug mode and listens at port 4000.

You can compile and run the JPrintStackTrace tool as shown in Figure 4.

Figure 4: Running JPrintStackTrace

```
javac -cp tools.jar;. JPrintStackTrace.java

java -classpath tools.jar;. JPrintStackTrace -r localhost
-p 4000 -e jpst.exclude -l jpst.txt
```

You can get help on how to use *JPrintStackTrace* by running the *help* tool without passing any command line options. However, there are four command line options that are available, as described below:

- -r <remote_host_name>: The name of the machine running the program to be debugged. In this case it is 'localhost'
- -p <port_no>: The port number at which the program to be debugged is listening. In our case it is '4000'
- -e <class_exclude_list_file> : List of Java classes that need to be excluded by the tool. A sample list of excluded classes is shown in Figure 5.
- -l <log_file> : Log file

Figure 5: A sample list of excluded classes

```
java.*
sun.*
javax.*
org.*
```

The JPrintStackTrace tool, once connected, will run the *Simple* program and listens for an exception event in the target JVM. On getting the exception event it will call the printStackTrace() method on the exception object at the target JVM to get a detailed stack trace as shown in Figure 6.

Figure 6: JPrintStackTrace in action

```
display14: nullStr: null
java.lang.NullPointerException
        at Simple.display14(Simple.java:7)
        at Simple.display13(Simple.java:15)
        at Simple.display12(Simple.java:23)
        at Simple.display1(Simple.java:31)
        at Simple.main(Simple.java:40)
java.lang.Throwable: display14
        at Simple.display14(Simple.java:9)
        at Simple.display13(Simple.java:15)
        at Simple.display12(Simple.java:23)
        at Simple.display1(Simple.java:31)
        at Simple.main(Simple.java:40)
java.lang.Throwable: display13
        at Simple.display13(Simple.java:17)
        at Simple.display12(Simple.java:23)
        at Simple.display1(Simple.java:31)
        at Simple.main(Simple.java:40)
java.lang.Throwable: display12
        at Simple.display12(Simple.java:25)
        at Simple.display1(Simple.java:31)
        at Simple.main(Simple.java:40)
java.lang.Throwable: display12
        at Simple.display12(Simple.java:25)
        at Simple.display1(Simple.java:31)
        at Simple.main(Simple.java:40)
```

Happy debugging

From the figure 6 you can clearly see the source code line that triggered the exception. It is line 7 in function display14() that triggered the exception. Compare the output shown in figure 2 and figure 6 and you will definitely appreciate the power of JPrintStackTrace tool. Remember that you get all this without making any changes to your source code!

REFERENCE

http://java.sun.com/products/jpda/index.jsp

END

By: Raja R.K. *The author is Project Leader at HCL Technologies Ltd, Chennai*